No.437

A Formulation of a Simulation Modelling Methodology

by

Ryo SATO

September 1990

A Formulation of a Simulation Modelling Methodology

Ryo SATO
Institute of Socio-Economic Planning,
University of Tsukuba,
Tsukuba city, Ibaraki 305, Japan

abstract

The aim of this paper is to provide a conceptual basis of modelling of discrete event systems, that is, a guideline along which making models and their programs and modification of them are carried out.

For this purpose we formulate the three phase approach that was proposed as a methodology of discrete event simulation. Using the methodology analysts model static interactions between entities in the target system and then make a program that shows dynamic aspects of the system. The formulation, based on the mathematical general systems theory, shows what determines the dynamics of a discrete event system resulted from the program, what information is used and how they are related in models in the methodology.

More concretely, a simulation program used in the three phase approach is formulated as a state space represenatation whose state space consists of queue and a set of internal records. Those records hold internally set occurrence time of activities in corresponding entities. And the role which activity interaction diagrams play as static skeleton models in the methodology is explicitly shown in terms of its simulation program.

1. Introduction

There are three issues in decision making based on simulation: modelling, programming, and statistics(Fishman 1973). This paper concerns modelling and programming.

By simulation we mean here that of queuing structure and that called discrete event simulation. Although calculation of a system of difference equations is sometimes called as simulation, we restrict the usage of the word. That kind of calculation should be taken as a discrete time system. Relation between discrete event systems and discrete time systems has not been much clear yet.

Pidd[2] has noticed that there are three types of errors on validation of simulation models after considering what is meant by validation concluding that a model can be said to be valid for particular purpose under specific assumptions. The first and second ones are well known in statistical hypothesis testing. A Type I error occurs when an valid model is wrongly rejected. An error of Type II occurs when an invalid model is taken to be valid. Much more severe is of Type Zero. It occurs modeler/tester asks the wrong questions, and then the model resulted is over-elaborated and/or over-simplified. It can be happen that a part of the model is too elaborated while other part has insufficient detail. To avid errors of this type Pidd[2] has proposed involvement of client of the study and the users of the result to use their knowledge through usage of non-technical language, diagrams and/or graphics to represent the problems and models. Furthermore he insisted that explicit and evolutionary approach to models and programs should be used because it is better to start a simple skeleton model than to have over-elaborated disaster that no one can understands.

To be realize the above idea Pidd[2] proposed a simple diagram, called an activity cycle diagram, for provision of non-technical representation of a model, and three phase simulation program which is supposed to be carry out the simulation according to the interaction of entities' behavior that is depicted in the diagram. (According to Pidd[2], activity cycle diagrams were popularized by Hills (Hills, P.R., *HOCUS*, P.E. Group, Egham, 1971) and three phase approach to discrete event simulation was first described by Tocher (Tocher, K.D., *The Art of Simulation*, English Universities Press, 1963)). Although it might seem to be easy to understand and do actually the methodology, by which one can start to make an activity cycle diagram and then build a corresponding program to get data of simulation experiments, several important issues are not much clarified. Relation between activity cycle diagrams and three phase simulation programs and the orientation along which we modify program to get a model of the intended situation have not made clear yet. Since there are many programs to do same calculation for a set of specific input, errors of Type Zero can be caused if there is no guideline to modify programs. System theoretic formulation provides a framework, by which we can

understand some meaning of simulation models and their programs, and then serves as a guideline for modification of them.

A typical simulation-based decision making process is shown in Fig. 1.

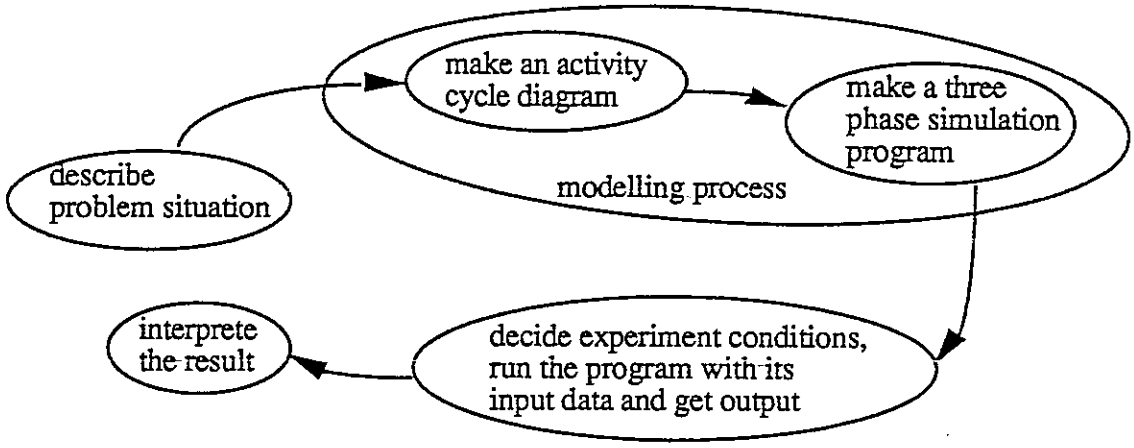

Fig. 1. Outline of simulation-based decision making

In this paper the methodology proposed by Pidd[2] as three phase approach, which concerns the modelling process in Fig.1, is formulated in terms of dynamics and interaction constructing a state space representation of a discrete event system is constructed. The formulation is to provide systems theoretical basis of simulation modelling and then to give insight for avoiding errors of Type Zero. The questions we would like to answer are:

(i) What is a discrete event system?

(ii) What information fully decides the dynamics of a discrete event system?
That is, in rigorous sense what kind of dynamics is dealt with and how?

(iii) How interactions are modelled and built into a program?

2. Basic Concepts

In this section systems, state space representations and some notation are defined according to Mesarovic and Takahara[1].

Definition 1. *system*

A system is a relation of an input set and output set. If those two sets are sets of time functions defined on the same time set, then the system is called a time system.

The value set of inputs of a time system is called input alphabet and that of output called output alphabet. Usually one of the set of non-negative real numbers R^+ or that of non-

negative integers Z^+ is taken as a time set. Let x be a function from a time set T to a input alphabet. For any t, t' ε T, t ≤ t', the restriction of domain of x to [t, t') is written as $x_{tt'}$. That is, $x_{tt'}(\tau) = x(\tau)$ for any τ, t ≤ τ < t'. Similarly x_t is defined as $x_t(\tau) = x(\tau)$ for any τ, t ≤ τ. Let x and x' be arbitrary functions from T to the same alphabet A. For any t ε T, we can define another element x": T → A by

$$x''(t'') = \begin{cases} x(t''), & \text{if } t'' < t \\ x'(t''), & \text{if } t'' \geq t \end{cases}$$

x" is called the concatenation of x_{0t} and x'_t and denoted by $x'' = x_{0t} \cdot x'_t$. We define input space X of S by $X = \{x \mid (\exists y)((x, y) \varepsilon S)\}$. The output space Y of S is defined similarly. The power set of a set A is denoted by P(A), that is, $P(A) = \{A' \mid A \supseteq A'\}$.

State space is a key concept by which we can grasp the behavior of the dynamic system.

Definition 2. *state space representation*

Let $S \subset X \times Y$ be a time system with output alphabet B. A pair $\langle \Phi, \mu \rangle$ where

$$\Phi = \{\phi_{tt'} \mid \phi_{tt'} : C \times X_{tt'} \rightarrow C \text{ and } t, t' \varepsilon T, t \leq t'\}$$

and $\mu : C \rightarrow B$

is a state space representation of S if and only if the following conditions are satisfied:

(i) the functions Φ satisfies the following

(α) $\phi_{tt''}(c, x_{tt''}) = \phi_{t't''}(\phi_{tt'}(c, x_{tt'}), x_{t't''})$, where $t \leq t'' \leq t'$ and $x_{tt''} = x_{tt'} \cdot x_{t't''}$

(β) $\phi_{tt}(c, x_{tt}) = c$

(ii) $(x,y) \varepsilon S$ if and only if there exists some $c \varepsilon C$ such that for any $t \varepsilon T$

$$y(t) = \mu(\phi_{0t}(c, x_{0t})).$$

C is called the state space of $\langle \Phi, \mu \rangle$.

Especially Φ is called a transition family if it satisfies (α) and (β) of the above definition.

State space representations are wide-spread framework to recognize dynamics of a time system in causal way. Mesarovic and Takahara[1] shows that a time system is causal if and only if it has a state space representation. Therefore what decides the dynamics of a system is equivalent to what information can be used as a state space.

3. Discrete Event System

The target of our study, discrete event system is defined as follows:

Definition 3. *discrete event system*

If a time system $S \subset X \times Y$ satisfies the following four conditions then is called a discrete event system:

1) $T=[0, T_{end})$, $T_{end} \in R^+$;

2) There is a set A' and the input alphabet A of input space X is the power set of A'. That is, $A = P(A')$;

3) For any $(x, y) \in S$ the following two conditions holds:

3-1) $\{t \mid x(t) \neq 0$, 0 is the empty set$\}$ is finite;

3-2) $F(y) = \{[t, t') \mid y(t) = y(t'')$ for any $t'', t \leq t'' < t'$, and $y(t) \neq y(t')\}$ is finite and $\cup F(y) = T$.

In the above definition the third character is representation of a "discrete event" system. Taking an element of the set A' as an activity, the second character shows parallel execution of some activities. The fourth character represents that if there happens no event then the corresponding output remains at the same value. In other words the concerned system has no variable that changes continuously with time. A discrete event system is specified by a sextuple (A', B, T, X, Y, S). When there is no confusion we simply call $S \subset X \times Y$ a discrete dynamical system.

For an input x the set $\{t \mid x(t) \neq 0$, 0 is the empty set$\}$ is denoted by event(x) and an element of event(x) is called an event of x. Since event(x) is finite for any input x, we can assume that $event(x) = \{t_1, t_2, ..., t_n\}$, $t_1 < t_2 < ... < t_n$ for some integer n. A function $nextevent : X \rightarrow \{T \rightarrow T\}$ is defined as

$$nextevent(x)(t) = \begin{cases} t_k, \text{ if } t_{k-1} \leq t < t_k \text{ for some } k,\ 1 \leq k \leq n \\ T_{end}, \text{ if } t_n \leq t \end{cases}$$

,where $t_0 = t$. The nextevent function shows the next event of x at t.

In order to establish a realization theory which can be directly implemented as a computer program for simulation of a discrete event system, we need the following transformation of inputs of discrete event systems.

Definition 4. *L (time-list representation)*

Let x be an input of a discrete event system (A', B, T, X, Y, S). Notice $x(t) \in P(A')$ holds for any $t \in T$.

Define $L: X \rightarrow \{T \rightarrow \{A' \rightarrow R^+\}\}$ as

$$L(x)(t)(b) = \begin{cases} T_{end}, \text{ if } x(t') \text{ does not include b for any } t',\ t \leq t' < T_{end}, \\ \min\{t' \mid x(t') \text{ includes b at } t',\ t' > t\}, \text{ otherwise} \end{cases}$$

for any $x \in X$, $t \in T$, and $b \in A'$. L(x) is called a time-list representation of x.

For any input function x, L(x) is called the time-list representation (or simply time-list) of x. The meaning of L(x) is simple. Any input x of any discrete event system has finite events(x) by definition 3. So event(x) can be written as $\{t_1, t_2, ..., t_n\}$, and without loss of generality $t_i < t_j$ holds if $i < j$. When we write L(x) as x^, x^(t) is a function from A' to R^+ for any time t. Let b be an event. Then the fact $x^(t)(b) = t_3$, for example, shows that at the time t the event b will occur at t_3. Since A' is finite, x^(t) can be seen as a table or a vector which tells each event will happen at each of recorded times.

The following lemma shows some relation between L(x) and events of x.

Lemma 1

Let x be an arbitrary input of a discrete event system and event(x) = $\{t_1, t_2, ..., t_n\}$, $t_1 < t_2 < ... < t_n$. Then for any k, $1 \le k-1 \le n-1$, the followings hold:

(i) $L(x)_{tt'}(\tau) = L(x)_{tt'}(t_{k-1})$

for any t, t', $t_{k-1} \le t < t' \le t_k$, and any τ, $t \le \tau < t'$.

(ii) $L(x)_{t_{k-1}t}(\tau)(b) \ge \text{nextevent}(x)(t_{k-1}) = t_k$

holds for any $b \in A'$, any t, $t > t_{k-1}$, and any τ, $t_k \le \tau < t'$. Furthermore there is a $b' \in A'$ such that $L(x)_{t_{k-1}t}(\tau)(b') = t_k$.

Proof. It is clear from the above definitions of L and nextevent. □

The function L has its inverse which is written as L^{-1}. For a discrete event system S whose input space is X, the time system whose input space is L(X) is written as L(S).

Definition 5. *discrete event dynamical system*

Let S be a discrete event system. A discrete event dynamical system of S is a state space representation of L(S).

The reason why the input of a state space representation of a discrete event system is L(X) instead of X, is that the formulation is in programming orientation. In Section 5 a simulation program will be formulated and its input is L(X). Since it is shown that the program is a state space representation and then we can get a concrete program of the dynamics. In other words this definition will provide us not only an analysis of dynamics of discrete event systems but also the way how to design and implement it as a program.

4. Activity Interaction Diagram

As depicted in Fig. 1, Pidd's approach to simulation modelling is two fold. The first is creation of diagrammatic skeleton model called an activity cycle diagram. The concept of activity cycle diagram is modified and then the name is changed as activity interaction diagram. The reason of the change is explained later.

Definition 6. *activity interaction diagram*

An tentuple (E, B_PE, B_BE, A, A_P, A_B, f_P, f_B, W, D) is called an activity interaction diagram if it satisfies following conditions;

1) E is a finite set named as entity set and its element is called entity. B_PE and B_BE are Psubsets of E.

2) A is a finite set named as activity set and its element is called activity.

3) W is a finite set named as wait set and its element is called a waiting queue.

4) Let $V=A \cup W$. The graph $D \subset V \times V$ satisfies the following three conditions:

4-1) For any $a \varepsilon A$, $(a, w) \varepsilon D$ holds for some $w \varepsilon W$.

4-2) $(s_1, s_2) \varepsilon D \rightarrow (s_1 \varepsilon A \ \& \ s_2 \varepsilon W)$ or $(s_1 \varepsilon W \ \& \ s_2 \varepsilon A)$: alternation of A and D.

4-3) D is finite.

5) $A_P = \{ a \varepsilon A \mid (w, a) \varepsilon D$ holds for any $w \varepsilon W \}$.

6) $A_B = \{ a \varepsilon A \mid \{ w \varepsilon W \mid (w, a) \varepsilon D$ holds$\}$is singleton$\}$.

7) Each element of A_P corresponds to exactly one element of B_PE and vise versa. This correspondence is denoted by f_P: $A_P \rightarrow B_PE$. The restriction of f_{EA} to B_PE coincides with f_P^{-1}. Also there is another one-to-one correspondence f_B from A_B to B_BE, and the restriction of f_{EA} to B_BE coincides with f_B^{-1}.

An activity interaction diagram is a graph Fig. 2 depicts an example of an activity interaction diagram of a fastfood restaurant. Customers arrive randomly and clerks serve them. If a customer wait for a long time then he will leave queue in frustration. $s_1 \rightarrow s_2$ in graph representation is written as (s_1, s_2) in definition 6. A path of arrows in accordance with their directions shows how an entity of an entity class behaves over time.

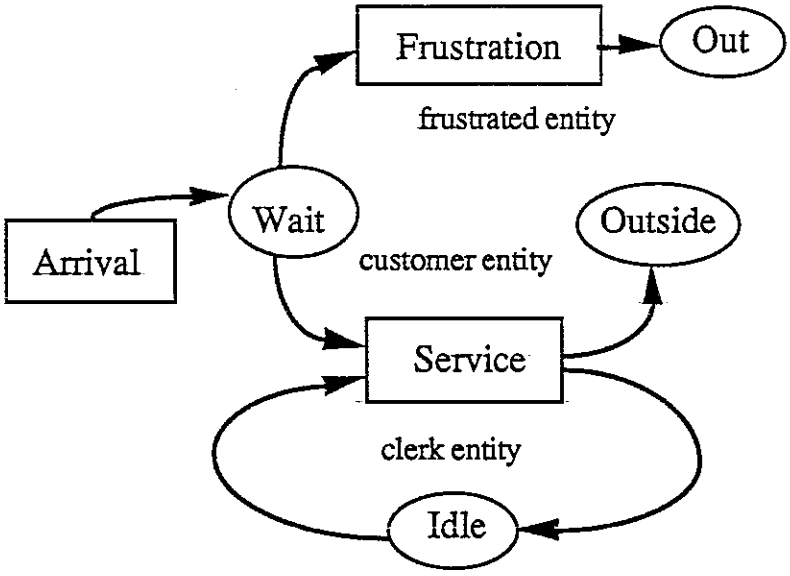

Fig. 2. activity interaction diagram

An activity is also called activity state and represented by a square, and a waiting queue called also wait state is by an ellipse. All of waiting queue are not real queue in a simulation program. Some of them are used just to represent conditions by which an activity can occur. In such case both 0 (or false) and 1 (or true) might be used as possible values of the queue. Before some activity starts the previous activity must be finished and an entity must be in a certain wait state. Thus activity state and wait state appear alternatively in any path.

An element of A_P is called B_P activity. It is an abbreviation for *pure bootstrapped activity*. Here "*bootstrapped*" means that the time of next occurrence is determined and recorded in the corresponding entity in a simulation program. And "*pure*" means that if the activity occurs then the next occurrence time is determined independently from the state of the system. In other words B_P activity can happens at any time. "Arrival" in Fig. 2 is an example of this type. B_B activity is an element of A_B. It is bootstrapped by a B_P activity. "Frustration" is an example because how long a customer stays in queue is determined when he arrives. An element of A, that is not in neither A_P nor A_B, is called C activity. In other words if an activity square has more than one arrow getting into then it is C activity. Its occurrence is *conditioned* by related queues in the program. An example of C activity is "Service" in Fig. 2.

Activity interaction diagrams are slightly different from activity cycle diagrams which is defined in Pidd[2]. Firstly the name is different, and secondly any possible path with respect to an entity may not make closed cycle in activity interaction diagram while activity cycle diagram urges to do that.

The name is changed because one of the main information represented by the diagram is interactions between entities' behavior as Pidd himself said. Making cycle has no effect in making a corresponding simulation program and no conceptual value. Precisely speaking,

interaction between entities is depicted through a typical (or representative) entity of the same character, that is, through classes of entities. The interaction suggests the existence of conditions on mutual possible behavior of entities of different classes of entities at any time.

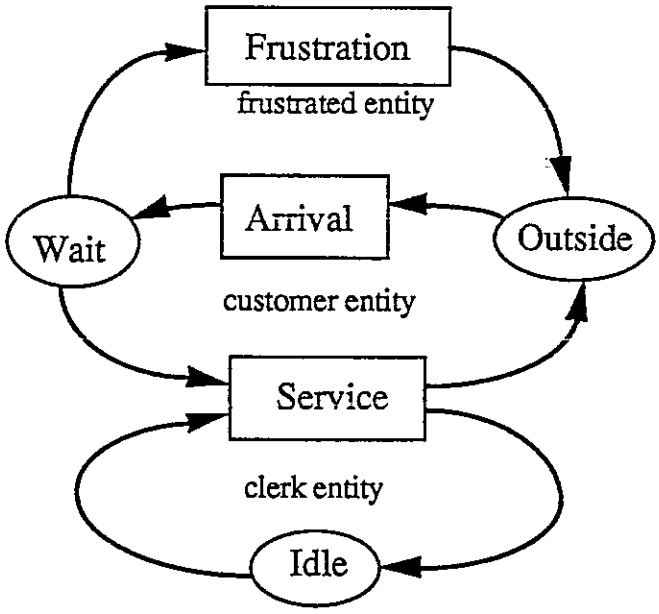

Fig. 3. activity cycle diagram

Although both activity cycle diagrams and activity interaction diagrams have the same philosophy, the former urged to make any path cycle. For example, the same interaction of clerk and customer in a fast food restaurant depicted in Fig.2 is shown as activity cycle diagram in Fig.3. In modelling a problem situation by activity interaction diagram modelers think about history of not a class of entities, but a typical entity, like a customer or a clerk. An arrow from "Outside" to "Arrival" not only does not be needed but also can be misleading because a customer entity may not come again to the window after once got the service. Furthermore other customers can arrive while the previously arrived customer is still in queue, not outside. Thus when there are activities, like arrival, that occur independently from other status of the system, cycle representation does not seem to be suitable.

The above two kind of consideration make us change the name and definition of activity cycle diagram as activity interaction diagram.

An activity interaction diagram can correspond to discrete event system. One of them is defined below.

Definition 7. *relevant discrete event system*

Let (A', B, T, X, Y, S) be a discrete event system. It is called a relevant discrete event system with an activity interaction diagram (E, B_PE, B_BE, A, A_P, A_B, f_P, f_B, W, D) if satisfies the following conditions:

1) $A' = A_P$.

2) The alphabet B of output Y is a subset of functions $\{f \mid f:W \rightarrow Z^+\}$.

5. Three Phase Simulation System

In the following definition a framework of simulation program based on the three phase approach[2] is formulated. And it will be proved to be a state space representation (discrete event dynamical system) of a discrete event system. In the following /*...*/ is a comment.

Definition 8. *three phase simulation system* $<\Phi, \Pi_Q>$

8-1) *definition of auxiliary sets*

Clock = $[0, T_{end}]$, where $T_{end} \varepsilon R^+$ /* time set of three phase simulation system */

$\Delta \varepsilon$ Clock /* The smallest time-slice to proceed simulation */

Entity = {1, 2, ..., n} /*Every entity has a unique name as a number*/

Entity = B_PEntity $\cup$ B_BEntity $\cup$ B_CEntity

, where B_PEntity, B_BEntity and B_CEntity are mutually disjoint.

AnEntityState = TimeCell×NextActivity×Avail

,where TimeCell = Clock

NextActivity = B_PActivity $\cup$ B_BActivity $\cup$ B_CActivity

Avail = {available, void} /*shows whether the time cell of the entity state is currently available or not*/

, where B_PActivity, B_BActivity and B_CActivity are finite and mutually disjoint.

/* Each entity is associated to an activity. If the activity is an element of Activity and the value of Avail is "available" then the activity will occur at the time represented in TimeCell. If the activity is of CActivity possibility of occurrence of it is examined at each time any activity occurred and executed in C_Phase below. */

Activity = NextActivity $\cup$ CActivity

, where CActivity = $\{c_1, c_2, ..., c_w\}$ is finite and disjoint from NextActivity.

EntityStates: Entity $\rightarrow$ AnEntityState /* each entity has its state */

B_BEntityStates is the restriction of EntityStates to B_BEntity.

B_CEntityStates is the restriction of EntityStates to B_CEntity.

CEntityStates = B_BEntityStates $\cup$ B_CEntityStates

QueueId = {1, 2, ..., u} /* each queue in the system has its name as a number */

Queue: QueueId $\rightarrow$ Number /*Queue shows the length of a queueid */

f_{AfectQ}: Activity $\rightarrow$ P(QueueId) /* Each activity affects a queue specified by f_{AfectQ}. */

X = {x | x : Clock $\rightarrow$ P(B_PActivity), {t | x(t)≠0, 0 is the empty set} is finite} /* input */

BTL = {f | f : B_PActivity $\rightarrow$ [0, T_{end}]} /*Note that L(x)(t) is an element of BTL for any t ε [0, T_{end}] */

f_{NextB_BAct}: B_PActivity $\rightarrow$ B_BActivity, one-to-one mapping.

/* Each B_B activity is bootstrapped by its corresponding B_P activity, specified by f_{NextB_BAct}.*/

f_{BEA}: B_BEntity $\rightarrow$ B_BActivity ; one-to-one mapping.

f_{PEA}: B_PEntity $\rightarrow$ B_PActivity ; one-to-one mapping.

/* Occurrence time of B_P and B_B activities are set in its corresponding a B_P and B_B entities, respectively. Functions f_{PEA} and f_{BEA} specifies the correspondence.*/

f_{EE}: B_PEntity $\rightarrow$ B_BEntity, $f_{EE} = f_{BEA}^{-1} \cdot f_{NextB_BAct} \cdot f_{PEA}$.

f_{B_BEA} : B_BEntity $\rightarrow$ B_PActivity, $f_{B_BEA} = f_{NextB_BAct}^{-1} \cdot f_{BEA}$.

f_{NextB_CAct}: CActivity $\rightarrow$ B_CActivity : one-to-one mapping.

/* Each B_C activity is bootstrapped by its corresponding C activity, specified by f_{NextB_CAct}. */

/* If z is an element of a Cartesian product and has A-coordinate then the A-coordinate of z is written as "z.A". For example, if z = (a, b, c) ε A×B×C then z.A = a and z.B = b. Changes with time in EntityStates and Queue can be divided into three phase, each of which is named A_Phase, B_Phase, and C_Phase. */

8-2) *Transition in A_Phase*

/* The transition of this phase is time scan and characterized by functions f_{Scan} and f_{Dues}. */

f_{Scan}: EntityStates $\rightarrow$ Clock

f_{Scan}(entitystates) = min {k | k = entitystates(i).TimeCell and entitystates(i).Avail = available for some entity i ε Entity}

f_{Dues}: EntityStates $\rightarrow$ P(Entity), where P(Entity) is the set of subsets of Entity.

f_{Dues}(entitystates) =

{i | f_{Scan}(entitystates) = entitystates(i).TimeCell, and entitystates(i).Avail = available}

8-3) *Transition in B_Phase*

/* The transition of B_Phase is characterized by functions f_{AfectQ}, f_{B_Ent}, and f_{B_Que}. */

We define f_{B_Ent} as follows.

Let entitystates ε EntityStates be arbitrary and nextdues = f_{Dues}(entitystates).

f_{B_Ent} : EntityStates → EntityStates

f_{B_Ent}(entitystates) = entitystates' such that for each i ε Entity

case 1: if i ε nextdues ∩ (B_BEntity ∪ B_CEntity) then

entitystates'(i).NextActivity = entitystates(i).NextActivity /*unchanged*/

entitystates'(i).TimeCell = entitystates(i).TimeCell /* unchanged*/

entitystates'(i).Avail = void.

case 2: if i ε nextdues ∩ B_PEntity then

entitystates'(f_{EE}(i)).NextActivity = f_{NextB_BAct}(entitystates(i).NextActivity)

entitystates'(f_{EE}(i)).TimeCell =$f_{NextTime}$(c)(f_{NextB_BAct}(entitystates(i).NextActivity))

entitystates'(f_{EE}(i)).Avail = available.

case 3: when i is not in nextdues:

entitystates'(i) = entitystates(i) /* unchanged*/

, where $f_{NextTime}$: Clock×(B_BActivity ∪ B_CActivity) → Clock and

$f_{NextTime}$(c)(f_{NextB_BAct}(entitystates(i).NextActivity)) ≥ Δ + c holds.

f_{B_Que}: EntityStates×Queue → Queue

/* f_{B_Que} calculates Queue when any of Activity occurs. */

f_{B_Que}(e, q) = q_{nu}, such that q_{11} = q and

f_{QueVal}(e(i).NextActivity, $q_{i-1,u}$),

if i ε nextdues, k ε f_{AfectQ}(e(i).NextActivity) and k = 1;

q_{ij}(k) = f_{QueVal}(e(i).NextActivity, $q_{i,j-1}$),

if i ε nextdues, k ε f_{AfectQ}(e(i).NextActivity), k = j and 1 < k ≤ u;

$q_{i,j-1}$(k), otherwise;

for each i and j, 1 ≤ i ≤ n, 1 ≤ j ≤ u, where f_{QueVal}:Activity×Queue → Number.

/* Each activity is associated to queues specified by f_{AfectQ}. If an activity is supposed to take place the associated queues are calculated by f_{B_Que}. Whether an activity occurs or not at the time c is decided from the fact that the activity is an element of nextdues. */

8-4) *transition in C_Phase*

$f_{C\_condition}$: Queue → P(CActivity)

f_{CB}: CActivity → B_CEntity : one-to-one mapping.

Let i ε Entity, queue ε Queue and entitystates ε EntityStates be arbitrary and c = f_{Scan}(entitystates).

f_{C_Ent}:Queue×EntityStates → EntityStates
f_{C_Ent}(queue, entitystates) = entitystates' such that for any i ε Entity
case 1: if i ∉ f_{CB}($f_{C\_condition}$(queue)) or $f_{C\_condition}$(queue) is empty then
entitystates'(i) = entitystates(i) /* unchanged */
case 2: if i ε f_{CB}($f_{C\_condition}$(queue)) then
entitystates'(i).NextActivity = f_{NextB_CAct}(f_{CB}^{-1}(i))
entitystates'(i).TimeCell = $f_{NextTime}$(c)(f_{NextB_CAct}(f_{CB}^{-1}(i))),
,where $f_{NextTime}$(c)(f_{NextB_CAct}(f_{CB}^{-1}(i))) ≥ Δ + c holds,
entitystates'(i).Avail = available

/* B_CActivity is another type of bootstrapped activity. In Fig. 2 "Service" is a C activity. When we think the length of service time varies to each customer, the time of finishing service is determined when the service starts. Thus the activity of the end of service occurs at the set time. A type of activities such as the end of service that is bootstrapped by a C activity is called B_CActivity. The occurrence time stored in a B_CEntity. */

f_{C_Que}: Queue → Queue /* f_{C_Que} calculates Queue when any of CActivity occurs. */
$f_{C\_Que}(q) = q_{wu}$ such that $q_{11} = q$ and

$$q_{ij}(k) = \begin{cases} f_{QueVal}(c_i, q_{i-1,u}), \text{ if } c_i \varepsilon f_{C\_condition}(q_{i,j-1}), k \varepsilon f_{AfectQ}(c_i) \text{ and } k = 1; \\ f_{QueVal}(c_i, q_{i,j-1}), \text{ if } c_i \varepsilon f_{C\_condition}(q_{i,j-1}), k \varepsilon f_{AfectQ}(c_i), k = j \text{ and } 1<k\leq u; \\ q_{i,j-1}(k), \text{ otherwise} \end{cases}$$

for any i, j, $1 \leq i \leq w$, $1 \leq j \leq u$.

8-5) *construction of* $<\phi, \Pi_Q>$

f_B:Queue×EntityStates → Queue×EntityStates.
Let q ε Queue and e ε EntityStates be arbitrary.
$f_B(q, e) = (f_{B\_Que}(e, q), f_{B\_Ent}(e))$. /* f_{Scan} and f_{Dues} are used in calculation of f_B. */
/* f_C is a total function which satisfies the following */
f_C : Queue×EntityStates → Queue×EntityStates

$$f_C(q, e) = \begin{cases} f_C(f_{C\_Que}(q), f_{C\_Ent}(q, e)), \text{ if } f_{C\_condition}(q) \neq 0 \\ (q, e), \text{ if } f_{C\_condition}(q) = 0 \end{cases}$$

δ:Queue×EntityStates → Queue×EntityStates

$\delta = f_C \cdot f_B$

Let $e^B \varepsilon B_P$EntityStates and $e^C \varepsilon$ CEntityStates be arbitrary. We will identify a pair (e^B, e^C) as an element of EntityStates in natural way.

δ_C: Queue×EntityStates → Queue×CEntityStates is defined by $\delta_C(q, e) = (q', e'^C)$ such that $\delta(q, e) = (q', e'^B, e'^C)$ for some $e'^B \varepsilon B_P$EntityStates.

f_{XEnt}:BTL → B_PEntityStates

$f_{XEnt}(\beta) = e'$ such that

e'(i).NextActivity = $f_{PEA}(i)$,

e'(i).TimeCell = $\beta(f_{PEA}(i))$,

e'(i).Avail = available

for any i ε B_PEntity.

$\phi_{tt'}$: Queue×CEntityStates×$L(X)_{tt'}$ → Queue×CEntityStates is defined by as follows:

for any t and t', t < t',

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \begin{cases} (q_m, e^C{}_m), \text{ if } t_m \le t' \text{ and } t_{m+1} > t' \\ (q, e^C), \text{ otherwise,} \end{cases}$$

where $t_0 = t$, $(q_0, e^C{}_0) = (q, e^C)$, $t_k = f_{Scan}(f_{XEnt}(L(x)_{tt'}(t_{k-1})), e^C{}_{k-1})$ and $(q_k, e^C{}_k) = \delta_C(q_{k-1}, f_{XEnt}(L(x)_{tt'}(t_{k-1})), e^C{}_{k-1})$ for some positive integer n and each k, $1 \le k \le m+1$. Also define $\phi_{tt}(q, e^C, L(x)_{tt}) = (q, e^C)$ for any t.

The family of functions defined above $<\Phi, \Pi_Q>$, where $\Phi = \{\phi_{tt'} \mid t, t' \varepsilon T, t \le t'\}$ and Π_Q is the projection on Queue×CEntityStates along Queue, is called a three phase simulation system.

Well-definedness of $\phi_{tt'}$ must be shown. Since event(x) is finite we can assume that event(x) = $\{t_1, t_2, ..., t_p\}$, $t_1 < t_2 < ... < t_p$ for some integer p. At each t_i, $1 \le i \le p$, for some of CEntity, say j ε CEntity, CEntityStates(j).TimeCell is set by f_{B_Ent} and f_{C_Ent}, and for some of B_PEntity k B_PEntityStates(k).TimeCell by f_{XEnt}. Therefore if the number of Entity is r then at most r entities are set in EntityStates such that the state value of Queue×EntityStates changes at each of the set times. Thus the number of times that give is at most p*r even for t = 0 and t' = T_{end}.

In this paper we have restricted our consideration in the case where f_C is a total function, that is, the expansion of f_C is eventually stops by $f_{C\_condition}$ value being empty. In terms of computer program of three phase simulation the assumption that f_C is total requires that the program always stops.

For a three phase simulation system $<\Phi, \Pi_Q>$ the *resultant system* that $<\Phi, \Pi_Q>$ defines is denoted by Res($<\Phi, \Pi_Q>$), that is, Res($<\Phi, \Pi_Q>$) = $\{(L(x),y) \mid (\exists((q, e^C))(\forall t)\ (y(t) = \Pi_Q \cdot \phi_{0t}(q, e^C, L(x)_{0t}))\}$. Similarly the *state system* of Res($<\Phi, \Pi_Q>$) for a pair $<\Phi, \Pi_Q>$ is

defined as Res(<ϕ, I_{QE}>), where I_{QE} is the identity function on Queue×EntityStates, such that (L(x),y) ε Res(<ϕ, I_{QE}>) iff there is some (q, e^C) ε Queue×CEntityStates and $y(t) = \phi_{0t}$ (q, e^C, $L(x)_{0t}$) holds for any t.

In the following that ϕ is a transition family will be shown. Before that we need the following

Lemma 2

Let <ϕ, Π_Q> be a three phase simulation system. Then

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \begin{cases} \phi_{t''t'}(\delta_C(q, f_{XEnt}(L(x)_{tt'}(t)), e^C), L(x)_{t''t'}), & \text{if } t''=f_{Scan}(f_{XEnt}(L(x)_{tt'}(t)), e^C) < t', \\ (q, e^C), & \text{otherwise.} \end{cases}$$

Proof. Under the same notation in definition of $\phi_{tt'}$, assume $t_m \le t'$ and $t_{m+1} > t'$ holds. Since $f_{Scan}(f_{XEnt}(L(x)_{tt'}(t)), e^C) = t_1$ and $t_1 < t'$ holds we have to show

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \phi_{t_1t'}(\delta_C(q, f_{XEnt}(L(x)_{tt'}(t)), e^C), L(x)_{t_1t'}).$$

By definition $\phi_{tt'}(q, e^C, L(x)_{tt'}) = (q_m, e^C_m)$.

Since the definition of $\phi_{tt'}$ shows $\delta_C(q, f_{XEnt}(L(x)_{tt'}(t)), e^C) = (q_1, e^C_1)$ holds, and then

$$\phi_{t_1t'}(q_1, e^C_1, L(x)_{t_1t'}) = (q_m, e^C_m)$$

holds. This concludes the proof. □

Proposition 1

Let <ϕ, Π_Q> be a three phase simulation system. ϕ is a transition family.

Proof. Let q ε Queue, t, t' ε T, $t \le t'$, e^C ε CEntityStates and x ε X be arbitrary. It will suffice to show that

(*) $$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \phi_{t''t'}(\phi_{tt''}(q, e^C, L(x)_{tt''}), L(x)_{t''t'})$$

holds for any t'', $t \le t'' \le t'$, where $L(x)_{tt'} = L(x)_{tt''} \bullet L(x)_{t''t'}$.

For any k, $1 \le k \le m$, define $(q_0, e^C_0) = (q, e^C)$, $t_0 = t$, $t_k = f_{Scan}(f_{XEnt}(L(x)_{tt'}(t_{k-1})), e^C_{k-1})$ and $(q_k, e^C_k) = \delta_C(q_{k-1}, f_{XEnt}(L(x)_{tt'}(t_{k-1})), e^C_{k-1})$. We can assume that $t_m \le t'$ and $f_{Scan}(f_{XEnt}(L(x)_{tt'}(t_m)), e^C_m) > t'$ hold without loss of generality. Then we have $t = t_0 < t_1 < \ldots < t_m < t'$. By lemma 2 we have

(**) $$\phi_{t_{k-1}t'}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t'}) = \phi_{t_kt'}(\delta_C(q_{k-1}, f_{XEnt}(L(x)_{t_{k-1}t'}(t_{k-1})), e^C_{k-1}), L(x)_{t_kt'}) = \phi_{t_kt'}(q_k, e^C_k, L(x)_{t_kt'})$$

for any k, $1 \le k \le m$.

Let t'' be an arbitrary element of [t, t'). Firstly assume that $t'' = t_k$ for some k, $0 \le k \le m+1$. If $t'' = t_0$ then the equation (*) holds trivially. So assume $t'' = t_k$ holds for some k, $1 \le k \le m$. Then by applying (**) k times from t_0 we have

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \phi_{t_kt'}(q_k, e^C_k, L(x)_{t_kt'}).$$

In the same way, k-1 times application of (**), where t' is replaced by t_k, gives that

$$\phi_{tt_k}(q, e^C, L(x)_{tt_k}) = \phi_{t_{k-1}t_k}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t_k}).$$

Since $f_{XEnt}(L(x)_{t_{k-1}t_k}(t_{k-1})) = f_{XEnt}(L(x)_{tt'}(t_{k-1}))$ holds, we have

$\phi_{t_{k-1}t_k}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t_k}) = \phi_{t_kt_k}(q_k, e^C_k, L(x)_{t_kt_k}) = (q_k, e^C_k)$. Above all we have

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \phi_{t_kt'}(\phi_{tt_k}(q, e^C, L(x)_{tt_k}), L(x)_{t_kt'})$$
$$= \phi_{t''t'}(\phi_{tt''}(q, e^C, L(x)_{tt''}), L(x)_{t''t'})$$

, which is to be proved.

Secondly assume that $t_{k-1} < t'' < t_k$ holds for some k, $1 \le k \le m+1$. By (**) we have

$$\phi_{tt''}(q, e^C, L(x)_{tt''}) = \phi_{t_{k-1}t''}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t''}).$$

Since $t_k > t''$, $\phi_{t_{k-1}t''}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t''}) = (q_{k-1}, e^C_{k-1})$ holds. That is, $\phi_{tt''}(q, e^C, L(x)_{tt''}) = (q_{k-1}, e^C_{k-1})$. k-1 times application of (**) from t_0 gives that

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \phi_{t_{k-1}t'}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t'}).$$

Therefore if $\phi_{t_{k-1}t'}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t'}) = \phi_{t''t'}(q_{k-1}, e^C_{k-1}, L(x)_{t''t'})$ holds then we have

$$\phi_{tt'}(q, e^C, L(x)_{tt'}) = \phi_{t''t'}(q_{k-1}, e^C_{k-1}, L(x)_{t''t'}) = \phi_{t''t'}(\phi_{tt''}(q, e^C, L(x)_{tt''}), L(x)_{t''t'})$$

,which concludes the proof. By lemma 1 we have that $L(x)_{t_{k-1}t'}(\sigma) = L(x)_{t''t'}(\tau) = L(x)_{tt'}(t_{k-1})$ for any σ, $t_{k-1} \le \sigma < t_k$, and any τ, $t'' \le \tau < t_k$, and then $f_{Scan}(f_{XEnt}(L(x)_{t''t'}(t'')), e^C_{k-1}) = t_k$. Thus we have

$$\phi_{t''t'}(q_{k-1}, e^C_{k-1}, L(x)_{t''t'}) = \phi_{t_kt'}(\delta_C(q_{k-1}, e^C_{k-1}, f_{XEnt}(L(x)_{t''t'}(t'')), L(x)_{t_kt'})$$
$$= \phi_{t_kt'}(q_k, e^C_k, L(x)_{t_kt'}).$$

The equation (**) says this is equal to $\phi_{t_{k-1}t'}(q_{k-1}, e^C_{k-1}, L(x)_{t_{k-1}t'})$. □

Proposition 2

Let $<\phi, \Pi_Q>$ be a three phase simulation system. Then $L^{-1}(Res(<\phi, \Pi_Q>))$ is a discrete event system.

Proof. Let $(x,y) \in L^{-1}(Res(<\phi, \Pi_Q>))$ be arbitrary. It will suffice to show that $F(y) = \{[t, t') \mid y(t) = y(t'')$ for any t'', $t \le t'' < t'$, and $y(t) \ne y(t')\}$ is finite and $\cup F(y) = [0, T_{end})$.

There is some $(q, e^C) \in Queue \times CEntityStates$ and $y(t) = \phi_{0t}(q, e^C, L(x)_{0t})$ holds for any t. By the definition of ϕ the output y possibly changes its value only at $f_{Scan}(f_{XEnt}(L(x)(\tau)), e'^C)$ for arbitrary $\tau \in T_{end}$ and $e'^C \in CEntityStates$. In the same way we showed the well-definedness of ϕ, if the number of elements of nextevent(x) is p and that of Entity r then the

number of times ,when the state value changes and then the output also possibly does, is at most p∗r. The fact that $\cup F(y) = [0, T_{end})$ clearly holds. □

The above two propositions provide one of the two main results.

Theorem 3

Let $<\Phi, \Pi_Q>$ be a three phase simulation system. Then $<\Phi, \Pi_Q>$ is a discrete event dynamical system of $L^{-1}(Res(<\Phi, \Pi_Q>))$.

The so-called discrete event simulation is carried out by a program whose input is a time function whose value is in P(Activity), the power set of Activity. Length of time between events are taken from appropriate distributions through sampling technique.

The whole simulation system works as Fig. 4. After initialization of EntityStates by assigning appropriate values, a cycle consists of A_Phase, B_Phase and C_Phase is repeated until clock exceeds the pre-determined time T_{end}. Every time an activity occurs its corresponding entitystate changes its TimeCell value by sampling from its assigned distribution. When simulation is over a pair (x, y) is provided. When much data is needed to statistical evaluation experiment design should be used, although we do not mention it in this paper.

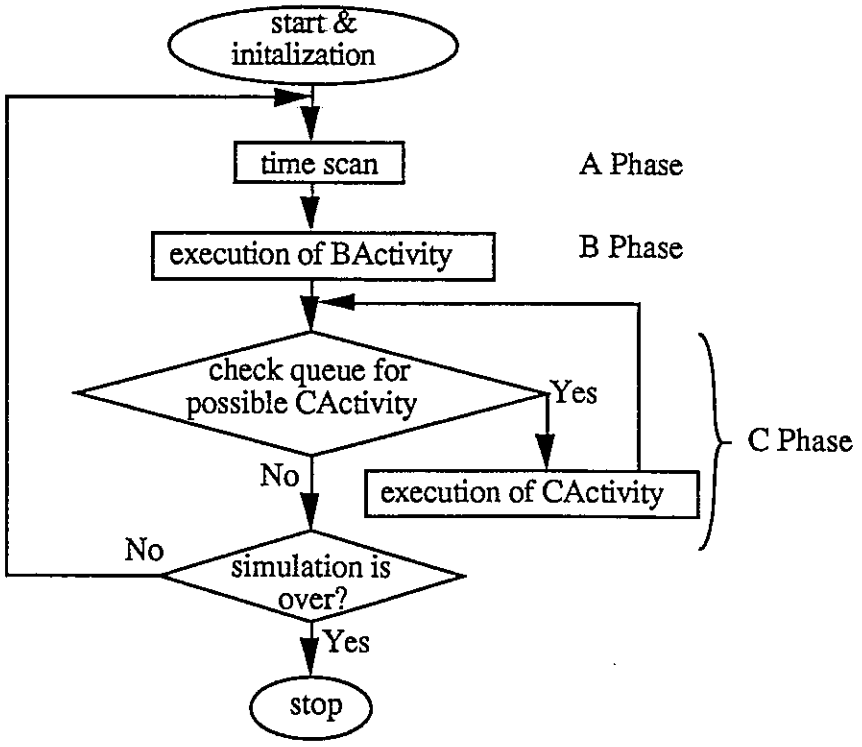

Fig. 4 . Execution of three phase simulation

Definition 9. *simulation-implementation of activity interaction diagram*

Let (E_{aid}, B_PE_{aid}, B_BE_{aid}, A_{aid}, A_{Paid}, A_{Baid}, f_{Paid}, f_{Baid}, W_{aid}, D_{aid}) be an activity interaction diagram and $<\Phi, \Pi_Q>$ a three phase simulation system. $<\Phi, \Pi_Q>$ is called a simulation-implementation of the activity interaction diagram if the following conditions hold:

1) E_{aid} = Entity, B_PE_{aid} = B_PEntity, B_BE_{aid} = B_BEntity

2) A_{Paid} = B_PActivity, A_{Baid} = B_BActivity, Activity $\supseteq A_{aid}$

3) $f_{Paid} = f_{PEA}^{-1}$, $f_{Baid} = f_{BEA}^{-1}$

4) W_{aid} = QueueId

5) For any $a \in A_{aid}$ and $w \in W_{aid}$ if $(a, w) \in D_{aid}$ or $(a, w) \in D_{aid}$ then $w \in f_{AfectQ}(a)$.

The following corollary shows a relation between simulation-implementations of an activity interaction diagram.

Corollary

Let $<\Phi, \Pi_Q>$ and $<\Phi', \Pi'_Q>$ be simulation-implementations of an activity interaction diagram. If $f_{NextB_BAct} = f'_{NextB_BAct}$, $f_{NextB_CAct} = f'_{NextB_CAct}$, $f_{NextTime} = f'_{NextTime}$, $f_{CB} = f'_{CB}$, $f_{C\_condition} = f'_{C\_condition}$, $f_{QueVal} = f'_{QueVal}$, and $f_{AfectQ} = f'_{AfectQ}$ hold, then $<\Phi, \Pi_Q> = <\Phi', \Pi'_Q>$.

Proof. By the definition of three phase simulation system if $f_{NextB_BAct} = f'_{NextB_BAct}$, $f_{NextB_CAct} = f'_{NextB_CAct}$, $f_{NextTime} = f'_{NextTime}$, $f_{CB} = f'_{CB}$, $f_{C\_condition} = f'_{C\_condition}$, $f_{QueVal} = f'_{QueVal}$, $f_{AfectQ} = f'_{AfectQ}$, $f_{PEA} = f'_{PEA}$, and $f_{BEA} = f'_{BEA}$ hold then we have $<\Phi, \Pi_Q> = <\Phi', \Pi'_Q>$. Since $<\Phi, \Pi_Q>$ and $<\Phi', \Pi'_Q>$ are simulation-implementations of an activity interaction diagram, $f_{PEA} = f'_{PEA}$ and $f_{BEA} = f'_{BEA}$ hold. □

Although the following theorem is easy to prove, it has conceptual importance to understand the three phase approach. It shows how a simulation implementation of an activity interaction diagram can be made and how the information that the diagram has is used in simulation.

Theorem 4

Let $<\Phi, \Pi_Q>$ be a simulation-implementation of an activity interaction diagram. Then $L^{-1}(Res(<\Phi, \Pi_Q>))$ is a relevant discrete event system with the diagram.

Proof. Trivial. □

6. Conclusion

What the formulation revealed are:

i) The dynamics of three phase simulation program is formulated as a state space representation whose state space is Queue×CEntityStates, and the resultant system is a discrete event system (Theorem 3). Realization theory of time systems, for example Mesarovic and Takahara[1], has shown that a state space has the information of the system by which further dynamics of the system is fully determined together with its input, and then that the sysytem is causal. The question of what information can be used as a state space, is not trivial but fundamentally important to answer what kind of system we will have by simulation.

ii) In the C phase of a three phase simulation system the condition check, that decides which C activities are ready to occur, should be examined about only the whole queue. There is no need to examine other information. This fact has not been exactly declared yet.

iii) The formulation of three phase simulation system (Definition 8) shows that the state Queue×CEntityStates is changed by activities as prescribed by f_B and f_C, and the priority of activities in execution of the system that can be occurred simultaneously is fixed.

iv) A skeleton model, activity interaction diagram, models entities, independent and conditioned activities, correspondence between activities and entities, queues and a condition which must be preserved in the simulation program of the skeleton model (Theorem 4). This fact gives exact meaning of the procedures for model-building in the three phase approach.

Generally speaking result of a program can be different if a part of program is changed. This rule holds true to discrete event system that is produced by a simulation program. Hence if one argue something about discrete event system and the system is result of a program, then he should also argue the consistency between his modelling and programming. In that situation, in addition to making and modifying simulation programs, the formulation can be used as a framework of argument. That is, it can be used as a guideline of discrete event simulation modelling.

Acknowledgement

The author is partially supported by the University of Tsukuba Project Research to which he would like to express his gratitude.

References

[1] M.D. Mesarovic, Y.Takahara: *Abstract Systems Theory*, (Lecture Notes in Control and Information Sciences 116), Springer, 1989.

[2] Michael Pidd: *Computer Simulation in Management Science* (Second edition), John Wiley & Sons, 1984.
[3] G.S.Fishman: *Concepts and Methods in Discrete Event Digital Simulation*, John Wiley & Sons, 1973.